Melt-pool dynamics -- system identification

1.Data Preparing

Read the data from txt file, retrieve timestamp, minor axix, laser power and velocity value

- Create data objects to represent data.
- Plot the data object.

load experiment data into workspace(time-domain data from a steady-state that laser power was perturbed from equilibrium values). --step test

load 2 or more experiment data --> one for system identification, the other for verification

display the content of the text file, to check if file can be read

```
% type('test_data.txt'); % melt-pool width
% type('power_data.txt');
% type('velocity_data.txt')
```

store all the data into a table (the names are changed for matlab format)

```
% experiment data 1
table_meltpool = readtable('Layer3.txt', ...
       'Delimiter', ',' ,'ReadVariableNames',true);
```

```
Warning: Table variable names were modified to make them valid MATLAB identifiers. The original names are
 saved in the VariableDescriptions property.
```

```
table_power = readtable('Layer3_power.txt', ...
       'Delimiter', ',' ,'ReadVariableNames',true);
```

```
Error using readtable (line 216)
Unable to open file 'Layer3_power.txt'.
```

```

% for experiment data 2
table_meltpool2 = readtable('Layer3.txt', ...
       'Delimiter', ',' ,'ReadVariableNames',true);
table_power2 = readtable('Layer3_power.txt', ...
       'Delimiter', ',' ,'ReadVariableNames',true);
```

There are 9 colums in total, extract the data according to their name

```
timeStamp = table_meltpool{:, 'field_header_stamp'};
minorAxis = table_meltpool{:, 'field_minor_axis'};
laserPower = table_power{:, 'field_power'};

% for experiment data 2
timeStamp2 = table_meltpool2{:, 'field_header_stamp'};
minorAxis2 = table_meltpool2{:, 'field_minor_axis'};
laserPower2 = table_power2{:, 'field_power'};
```

2. Plot data for simple visualization

Plot the melt-pool width (minor axis) and laser power agiainst timestamp

for open loop:

- **input: laser power**
- **output: melt-pool width**

Plotting the Input/Output Data

```
% for experiment data 1----------------------------------------
figure
yyaxis left
plot(timeStampe, laserPower);
yyaxis right
plot(timeStamp, minorAxis);
% add titles and lables
yyaxis left
title('Plots of Laser Power and Melt-pool width (20 layers)')
xlabel('TimeStamp')
ylabel('Melt-pool Width [pixels]')
yyaxis right
ylabel('Laser Power [Volt]')

legend('laser power','melt-pool width')

%xlim([1582856500000000000 1582856850000000000])
%ylim([0 250])

% for experiment data 2-------------------------------------------
figure
yyaxis left
plot(timeStampe2, laserPower2);
yyaxis right
plot(timeStamp2, minorAxis2);
% add titles and lables
yyaxis left
title('Plots of Laser Power and Melt-pool width (20 layers)')
xlabel('TimeStamp')
ylabel('Melt-pool Width [pixels]')

yyaxis right
ylabel('Laser Power [Volt]')

legend('laser power','melt-pool width')

%xlim([1582856500000000000 1582856850000000000])
%ylim([0 250])
```

Plot a single track (optional) -- just to see what it looks like

```
timeStamp_track= timeStamp(6036:6153);
minorAxis_track = minorAxis(6036:6153);
laserPower_track = laserPower(6036:6153);
figure
plot(timeStamp_track, minorAxis_track)

legend('minor axis')
xlabel('TimeStamp'), ylabel('Minor Axis [pixels]')
title('Response of Melt-pool Minor Axis (Single track)')
```

Identify Linear Models in the Command Line

Data Preparing

Estimate and validate linear models from single-input/single-output (SISO) data to find the one that best describes the system dynamics.

- Process data by removing offsets from the input and output signals.
- Estimate and validate linear models from the data.
- Simulate and predict model output.

Removing Equilibrium Values from the Data

the inputs and the outputs have nonzero equilibrium values, subtract the mean values from each signal: seek models around zero without modeling the absolute equilibrium levels in physical units.

```
%---------------from example-------------------------
%Input_exp1 = Input_exp1-...
%   ones(size(Input_exp1,1),1)*mean(Input_exp1(1:50,:));
%Output_exp1 = Output_exp1-...
%   mean(Output_exp1(1:50,:));
%Input_exp2 = Input_exp2-...
%   ones(size(Input_exp2,1),1)*mean(Input_exp2(1:50,:));
%Output_exp2 = Output_exp2-...
%   mean(Output_exp2(1:50,:));
%----------------------------------------------------
% experiment data 1
laserPower = laserPower - mean(laserPower(1:50,:)); % open loop input
```

```
Undefined function or variable 'laserPower'.
```

```
minorAxis = minorAxis - mean(minorAxis(1:50,:)); % open loop outloop
% experiment data 2
laserPower2 = laserPower2 - mean(laserPower2(1:50,:)); % open loop input
```

```
minorAxis2 = minorAxis2 - mean(minorAxis2(1:50,:)); % open loop outloop
```

Using Objects to Represent Data for System Identification

iddata***:*** Input-output data and its properties for system identification in the time or frequency domain

idfrd: Frequency-response data or model:

`syntax: data = iddata(y,u,Ts) --> create a data object`

```
Ts = 1/30; % Sampling time is 1/30s --> corresponds to sampling rate at 30 Hz, set by ROS publi
% ze = iddata(Output_exp1,Input_exp1,Ts); --example format
OpenLoop_system1 = iddata(minorAxis,laserPower,Ts); % for experiment1 data
OpenLoop_system2 = iddata(minorAxis2,laserPower2,Ts); % for experiment2 data
```

To view the properties of an `iddata` object,

```
get(OpenLoop_system1)
get(OpenLoop_system2)
```

To modify data properties, set units:

```
% Set time units to minutes
OpenLoop_system1.TimeUnit = 'sec';
% Set names of input channels
OpenLoop_system1.InputName = {'LaserPower'};
% Set units for input variables
OpenLoop_system1.InputUnit = {'Volt'};
% Set name of output channel
OpenLoop_system1.OutputName = 'MeltPoolWidth(MPW)';
% Set unit of output channel
OpenLoop_system1.OutputUnit = 'pixels';

% Set validation data properties, (validation data is experiment 2 data)
OpenLoop_system2.TimeUnit = 'sec';
OpenLoop_system2.InputName = {'LaserPower'};
OpenLoop_system2.InputUnit = {'Volt'};
OpenLoop_system2.OutputName = 'MeltPoolWidth(MPW)';
OpenLoop_system2.OutputUnit = 'pixels';
```

Plotting the Data in a Data Object

```
plot(OpenLoop_system1)
figure   % Open a new MATLAB Figure window
plot(OpenLoop_system2)
```

Selecting a Subset of the Data (to reduce time of calculation)

```
OpenLoop_system_subset1 = OpenLoop_system1(1:1000);
OpenLoop_system_subset2 = OpenLoop_system2(1:1000);
%Zv1 = OpenLoop_system(1:1000);
```

Estimating Impulse Response Models

description:

Estimate Step- and Frequency-Response Models

1. Frequency-response and step-response are *nonparametric* models that can help you understand the dynamic characteristics of your system.
2. estimate these models using the above mentioned data set.
3. The response plots from these models show the characteristics of the system: first order, second order or overdamped between input and output

Estimating the Frequency Response

three functions for estimating the frequency response, in system identification toolbox:

- `etfe` computes the empirical transfer function using Fourier analysis.
- `spa` estimates the transfer function using spectral analysis for a fixed frequency resolution.
- `spafdr` lets you specify a variable frequency resolution for estimating the frequency response.

Use **`spa`** to estimate the frequency response

```
Ge = spa(OpenLoop_system1);
```

Plot the frequency response as a Bode plot.

```
bode(Ge)
```

Estimating the Empirical Step Response

first estimate a non-parametric impulse response model (FIR filter) from

data and then plot its step response.

- impulseest: Nonparametric impulse response estimation
- N: Order of FIR model
- `sys = impulseest(data,N)` estimates an `N`th order impulse response model, corresponding to the time range 0 :*Ts* : (*N* –1)**Ts*, where `Ts` is the data sample time.

```
% model estimation
Mimp = impulseest(OpenLoop_system_subset1,60);
% step response
step(Mimp)
```

Estimating Delays in the System

reference: https://www.mathworks.com/help/ident/ug/model-structure-selection-determining-model-order-and-input-delay.html

Estimate Delays:

To identify parametric black-box models, you must specify the input/output delay as part of the model order.
(time in terms of the number of samples before the output responds to the input)

Estimating Delays Using the ARX Model Structure

Estimates the time delay in a dynamic system by estimating a low-order, discrete-time ARX model with a range of delays, and then choosing the delay that corresponding to the best fit.

To estimate the delay in this system, type: --> ***get the result in data samples*** (i.e. how many data samples the MPW is lag behind the laser power has changed), to get the actual time, you need to use sample number times sampling time.

```
delayest(OpenLoop_system1)
```

Estimating Model Orders Using an ARX Model Structure

Model order is one or more integers that define the complexity of the model.number of poles, the number of zeros, and the response delay.

estimate and compare simple polynomial (ARX) models for a range of model orders and delays, and select the best orders based on the quality of the model.

y(t) + a1y(t - 1) + … + anay(t - na) =

b1u(t - nk) + … + bnbu(t - nk - nb + 1) + e(t)

na is the number of poles, ***nb*** is the number of *b* parameters (equal to the number of zeros plus 1), the ***delay* or *dead time*** of the model is nk

Commands for Estimating the Model Order:

- `struc` creates a matrix of model-order combinations for a specified range of *na*, *nb*, and *nk* values. *nk*
- `arxstruc` takes the output from `struc`, systematically estimates an ARX model for each model order, and compares the model output to the measured output. `arxstruc` returns the *loss function* for each model, which is the normalized sum of squared prediction errors.
- `selstruc` takes the output from `arxstruc` and opens the ARX Model Structure Selection window, which resembles the following figure, to help you choose the model order.

To estimate model order for the first input/output combination:

1. Use `struc` to create a matrix of possible model orders.

make necessary asumptions based on above results:

na =1:5(We feel it might be a second orer because overshoot, so we create it in range from 1 to 5)

nb =1:5

nk =5 ---> needs to be changed accordingly (estimated delay time)

```
NN1 = struc(1:5,1:5,5);
```

2. Use `selstruc` to compute the loss functions for the ARX models with the orders in `NN1`

`OpenLoop_system(:,:,1)` selects the first input in the data if have multiple input, since we have only single input.

```
selstruc(arxstruc(OpenLoop_system1(:),OpenLoop_system1(:),NN1))
```

Estimating Transfer Functions

Specifying the Structure of the Transfer Function

from the ARX model estimatimation result, we have obtained he orders of the model and delay time.

estimate a transfer function of these orders using the `tfest` command.

- For the first input/output combination, use:
- Two poles, corresponding to `na=2` in the ARX model.
- Delay of 5, corresponding to `nk=5` samples (or 2.5 minutes) in the ARX model.

view a progress report by setting the `Display` option to on in the option

```
Opt = tfestOptions('Display','on');
```

Collect the model orders and delays into variables to pass to `tfest`.

```
%np = [2 1];
%ioDelay = [2.5 5];
np = 2;
ioDelay = 2.5; % in seconds
```

Estimate the transfer function.

```
mtf = tfest(OpenLoop_system_subset1,np,[],ioDelay,Opt);
```

View the model's coefficients.

```
mtf
```

Validating the Model

create a plot that compares the actual output and the model output using the `compare` command

```
compare(OpenLoop_system2,mtf)
```

Residual Analysis

evaluate the characteristics of the residuals.

`resid(Data,sys)` computes the 1-step-ahead prediction errors (residuals) for an identified model, and plots residual-input dynamics

```
resid(OpenLoop_system2,mtf)
```

Convert data to frequency domain. (optional)

```
OpenLoop_system_frequncy2 = fft(OpenLoop_system2);
```

Compute the residuals for identified model, `mtf`, and the frequency-domain data. Plot the residual response using red crosses.

```
resid(OpenLoop_system_frequncy2,mtf,'rx')
```

Estimate Process Model

Specifying the Structure of the Process Model

(*a low-order, continuous-time transfer function)*

Use the **`idproc`** command to create model structures, for input/output:

```
midproc0 = idproc({'P1D'}, 'TimeUnit', 'seconds');
```

`{'P2ZUD','P1D'}` specifies the model structure for each input/output combination:

`'P2ZUD'` represents a transfer function with two poles (`P2`), one zero (`Z`), underdamped (complex-conjugate)

poles (

`U`) and a delay (`D`).

`'P1D'` represents a transfer function with one pole (`P1`) and a delay (`D`).

Viewing the Model Structure and Parameter Values

```
midproc0
```

Specifying Initial Guesses for Time Delays

```
%----------------for 2 input 1 output system --------
%midproc0.Structure(1,1).Td.Value = 2.5;
%midproc0.Structure(1,2).Td.Value = 5;
%midproc0.Structure(1,1).Td.Maximum = 3;
%midproc0.Structure(1,2).Td.Maximum = 7;

%---------------for SISO system ---------------------
midproc0.Structure(1).Td.Value = 2.5;
midproc0.Structure(2).Td.Maximum = 3;
```

Estimating Model Parameters Using procest

`procest` is an *iterative* estimator of process models,

it uses an iterative nonlinear least-squares algorithm to minimize a cost function. e *cost function* is the weighted sum of the squares of the errors.

estimate parameters for linear continuous-time transfer-function

```
midproc = procest(OpenLoop_system_subset1,midproc0); %midproc0 as the model structure and OpenL
present(midproc)
```

Validating the Model

create a plot that compares the actual output and the model output.

```
compare(OpenLoop_system2,midproc)
```

perform residual analysis.

```
resid(OpenLoop_system2,midproc)
```

Change the algorithm for iterative parameter estimation to Levenberg-Marquardt.

```
Opt = procestOptions;
Opt.SearchMethod = 'lm';
midproc1 = procest(OpenLoop_system_subset1,midproc,Opt);
```

Estimating a Process Model with Noise Model

Including a noise model typically improves model prediction results but not necessarily the simulation results.

Use the following commands to specify a first-order ARMA noise:

```
Opt = procestOptions;
Opt.DisturbanceModel = 'ARMA1';
midproc2 = procest(OpenLoop_system_subset1,midproc0,Opt);
```

Compare the resulting model to the measured data.

```
compare(OpenLoop_system2,midproc2)
```

Perform residual analysis.

```
resid(OpenLoop_system2,midproc2)
```